Decrease and Conquer

Bahasan

- Defisini Decrease and Conquer
- 3 tipe Decrease and Conquer
- Insert sort
- Graph traversal
  - DFS
  - BFS
- Topological sort

Decrease-and-Conquer

1. **Kurangi instance masalah menjadi instance yang lebih kecil dari masalah yang sama**
2. **Memecahkan contoh yang lebih kecil**
3. **Perluas solusi dari instance yang lebih kecil untuk mendapatkan solusi ke instance asli**

- **Dapat diimplementasikan baik top-down atau bottom-up**
- **Juga disebut sebagai pendekatan induktif atau inkremental**

Decrease-and-Conquer

- ***Decrease and conquer*: metode desain algoritma dengan mereduksi persoalan menjadi beberapa sub- persoalan yang lebih kecil, tetapi selanjutnya hanya memproses satu sub-persoalan saja.**

- **Berbeda dengan *divide and conquer* yang memproses *semua* sub-persoalan dan menggabung semua solusi setiap sub-persoalan.**

Decrease-and-Conquer

- ***Decrease and conquer*** **terdiri dari dua tahapan:**

  1. Decrease**: mereduksi persoalan menjadi beberapa persoalan yang lebih kecil (biasanya dua sub- persoalan).**
  2. Conquer**: memproses satu sub-persoalan secara rekursif.**

- **Tidak ada tahap *combine* dalam *decrease and conquer.***

3 Tipe Decrease and Conquer

- ***Decrease by a constant*** **(usually by 1):**
  - **insertion sort**
  - **graph traversal algorithms (DFS and BFS)**
  - **topological sorting**
  - **algorithms for generating permutations, subsets**

- ***Decrease by a constant factor*** **(usually by half)**
  - **binary search and bisection method**
  - **exponentiation by squaring**
  - **multiplication à la russe**

- ***Variable-size decrease***
  - **Euclid's algorithm**
  - **selection by partition**
  - **Nim-like games**

What's the difference?

Pertimbangkan masalah eksponensial: Hitung x^n

- **Brute Force:** *n-1* multiplications
- **Divide and conquer:** $T(n) = 2*T(n/2) + 1$

  $= n-1$

- **Decrease by one:** $T(n) = T(n-1) + 1 = n-1$
- **Decrease by constant factor:** $T(n) = T(n/a) + a-1$

  $= (a-1) \log_a n$

  $= \log_2 n$ *when a = 2*

Insertion Sort

Mengurutkan array A[0..n-1], urutkan A[0..n-2] secara rekursif dan kemudian masukkan A[n-1] di tempat yang tepat di antara A[0..n-2] yang diurutkan

Biasanya diimplementasikan dari bawah ke atas (nonrekursif)

Contoh: Sort 6, 4, 1, 8, 5

6 | 4 1 8 5
4 6 | 1 8 5
1 4 6 | 8 5
1 4 6 8 | 5
1 4 5 6 8

Pseudocode Insertion Sort

ALGORITHM *InsertionSort*$(A[0..n-1])$

```
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
    v ← A[i]
    j ← i − 1
    while j ≥ 0 and A[j] > v do
        A[j + 1] ← A[j]
        j ← j − 1
    A[j + 1] ← v
```

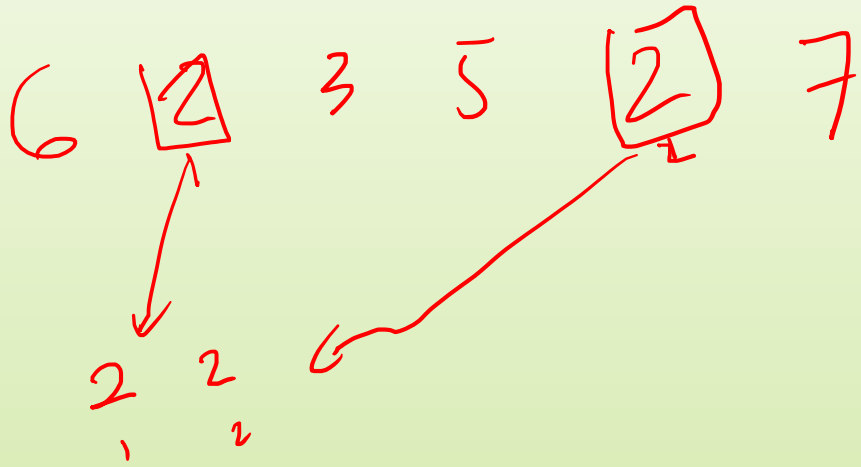
Analisis Insertion Sort

- **Time efficiency**

  $C_{worst}(n) = n(n\text{-}1)/2 \in \Theta(n^2)$

  $C_{avg}(n) \approx n^2/4 \in \Theta(n^2)$

  $C_{best}(n) = n - 1 \in \Theta(n)$

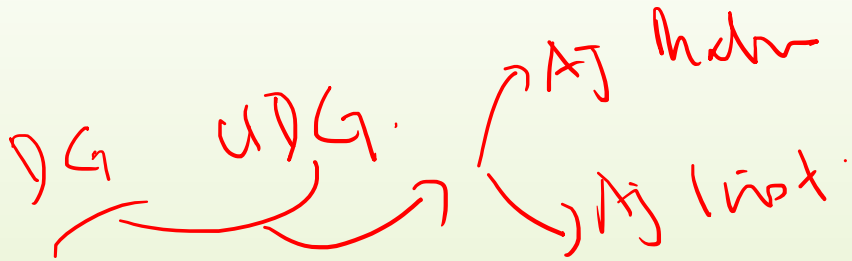


- **Space efficiency: in-place**

- **Stability: yes**

- **Best elementary sorting algorithm overall**

Graph Traversal



Banyak masalah memerlukan pemrosesan semua simpul (dan tepi) graf secara sistematis

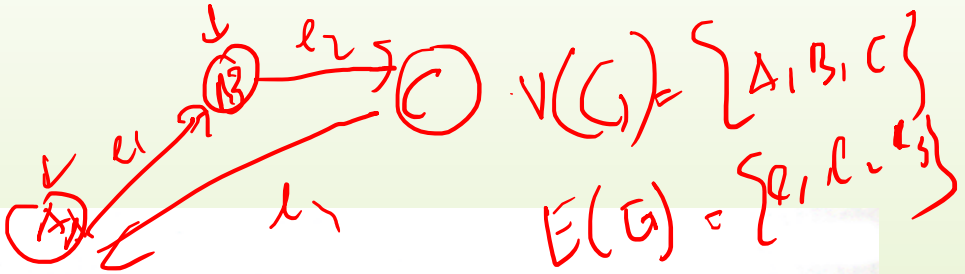
<u>Graph traversal algorithms</u>:

- **Depth-first search (DFS)**
- **Breadth-first search (BFS)**

Depth-First Search (DFS)

- **Mengunjungi simpul graf dengan selalu berpindah dari simpul terakhir yang dikunjungi ke simpul yang belum dikunjungi, mundur jika tidak ada simpul yang belum dikunjungi yang berdekatan.**
- **Rekursif atau menggunakan STACK**
  - **sebuah simpul didorong ke tumpukan ketika tercapai untuk pertama kalinya**
  - **sebuah simpul dikeluarkan dari tumpukan ketika menjadi jalan buntu, yaitu, ketika tidak ada simpul berdekatan yang belum dikunjungi**
- **Grafik "Menggambar ulang" dengan cara seperti pohon (dengan tepi pohon dan tepi belakang untuk grafik tidak berarah)**

Pseudocode DFS

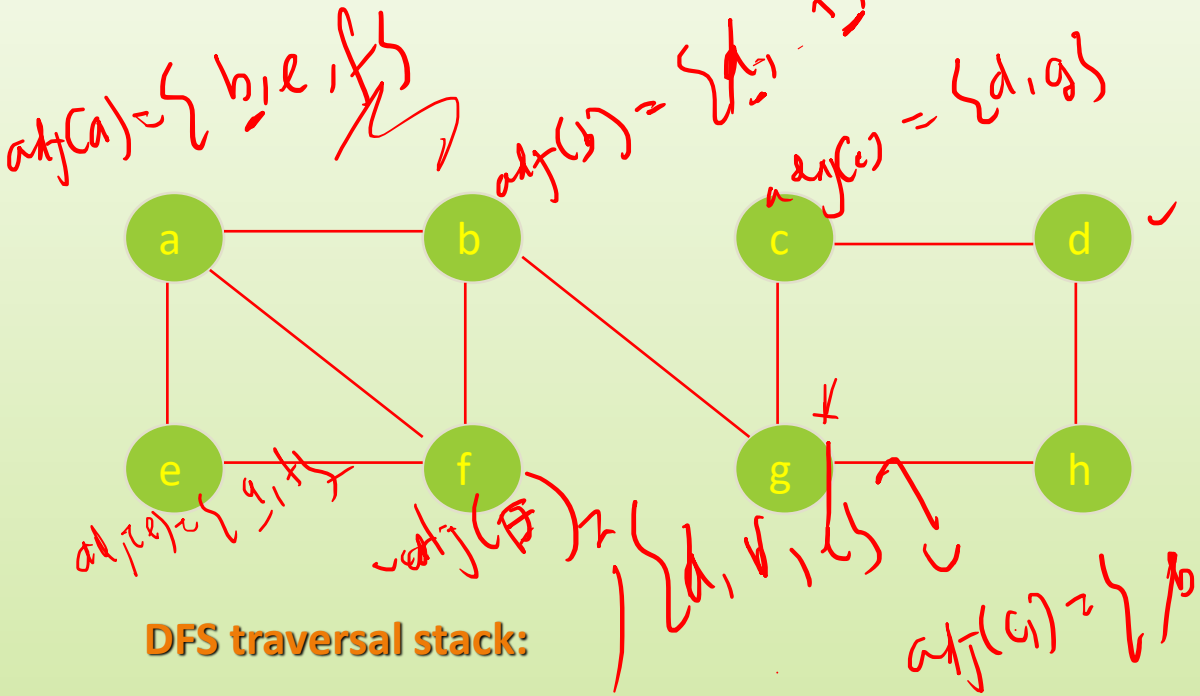


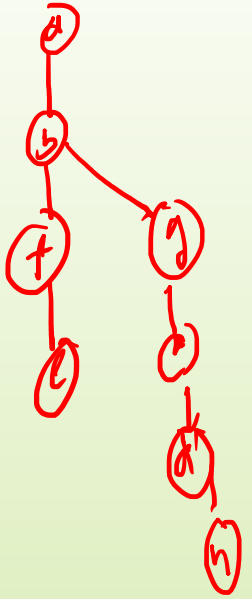
ALGORITHM *DFS*(G)

```
//Implements a depth-first search traversal of a given graph
//Input: Graph G = ⟨V, E⟩
//Output: Graph G with its vertices marked with consecutive integers
//in the order they've been first encountered by the DFS traversal
mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
        dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to vertex v by a path
//and numbers them in the order they are encountered
//via global variable count
count ← count + 1; mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```

Contoh: DFS traversal (undirected graph)

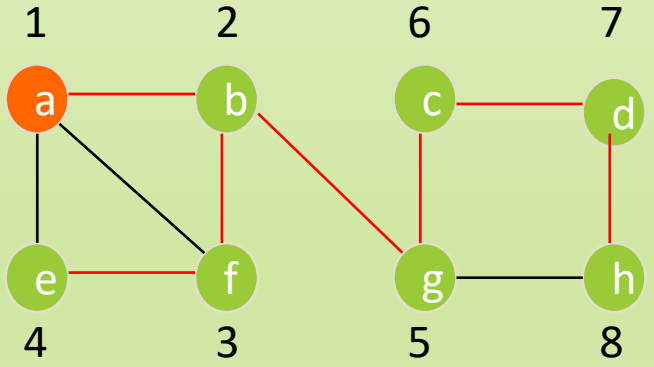




DFS tree:

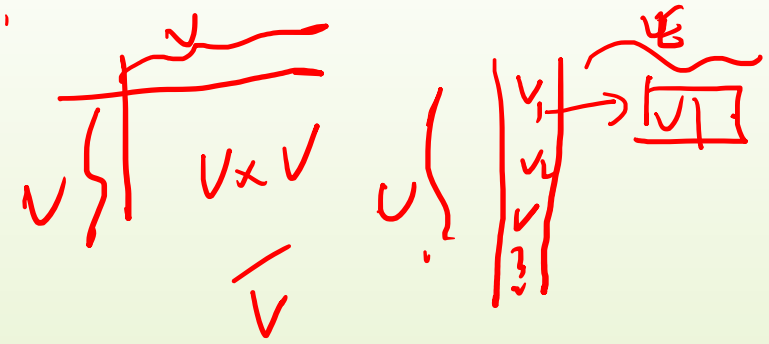
DFS traversal stack:

a ab abf abfe abf ab abg abgc abgcd abgcdh abgcd :



Red edges are tree edges and white edges are back edges.

Catatan tentang DFS

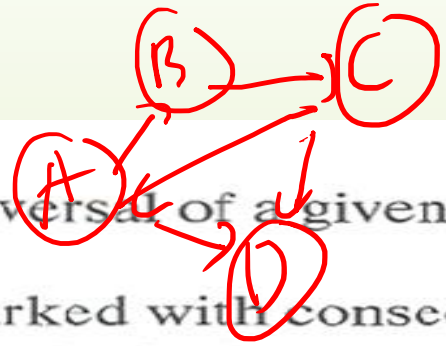


- **DFS dapat diimplementasikan dengan grafik yang direpresentasikan sebagai:**
  - **adjacency matrices: $\Theta(|V|^2)$. Why?**
  - **adjacency lists: $\Theta(|V|+|E|)$. Why?**
- **Menghasilkan dua urutan berbeda dari simpul:**
  - **urutan di mana simpul pertama kali ditemui (didorong ke tumpukan)**
  - **urutan di mana simpul menjadi buntu (muncul dari tumpukan)**
- **Aplikasi:**
  - **memeriksa konektivitas, menemukan komponen yang terhubung**
  - **memeriksa asiklikitas (jika tidak ada tepi belakang)**
  - **menemukan titik artikulasi dan komponen bikoneksi**
  - **mencari ruang keadaan masalah untuk solusi (dalam AI)**

Breadth-first search (BFS)

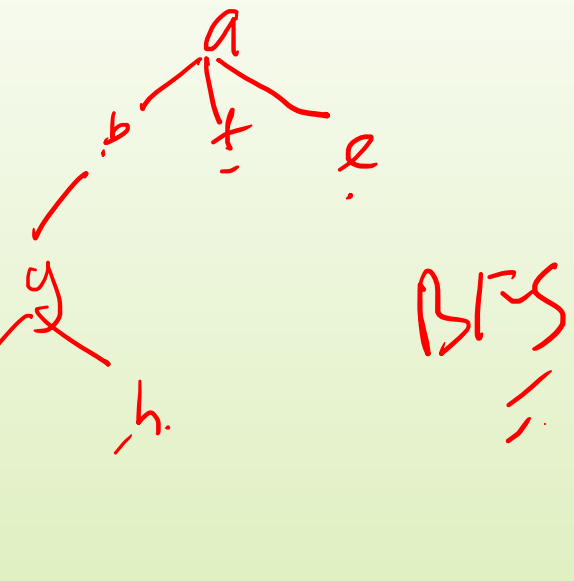
- **Mengunjungi simpul graf dengan berpindah ke semua tetangga dari simpul yang terakhir dikunjungi**
- **BFS menggunakan ANTRIAN**
- **Mirip dengan traversal pohon level demi level**
- **Grafik "Menggambar ulang" dengan cara seperti pohon (dengan tepi pohon dan tepi silang untuk grafik tidak berarah)**

Pseudocode of BFS

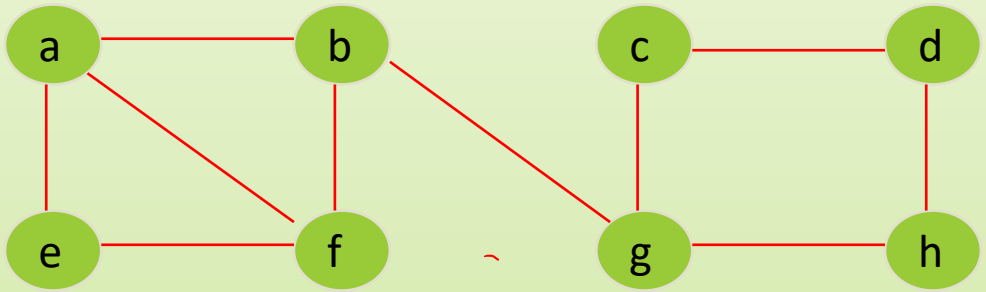


```
ALGORITHM BFS(G)
    //Implements a breadth-first search traversal of a given graph
    //Input: Graph G = ⟨V, E⟩
    //Output: Graph G with its vertices marked with consecutive integers
    //in the order they have been visited by the BFS traversal
    mark each vertex in V with 0 as a mark of being "unvisited"
    count ← 0
    for each vertex v in V do
        if v is marked with 0
            bfs(v)
    bfs(v)
    //visits all the unvisited vertices connected to vertex v by a path
    //and assigns them the numbers in the order they are visited
    //via global variable count
    count ← count + 1;   mark v with count and initialize a queue with v
    while the queue is not empty do
        for each vertex w in V adjacent to the front vertex do
            if w is marked with 0
                count ← count + 1;   mark w with count
                add w to the queue
        remove the front vertex from the queue
```

Contoh BFS traversal (undirected graph)

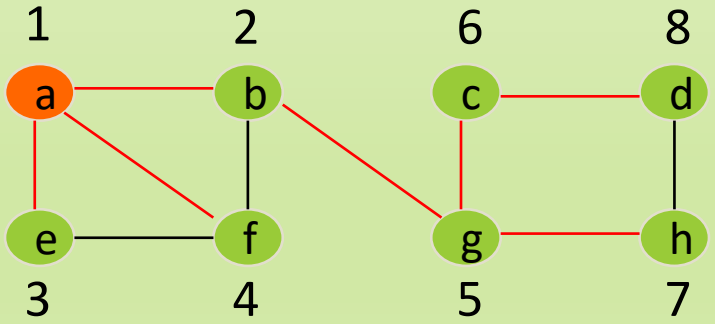


BFS traversal queue:



a bef efg fgg gg ch hd d

BFS tree:



Red edges are tree edges and white edges are cross edges.

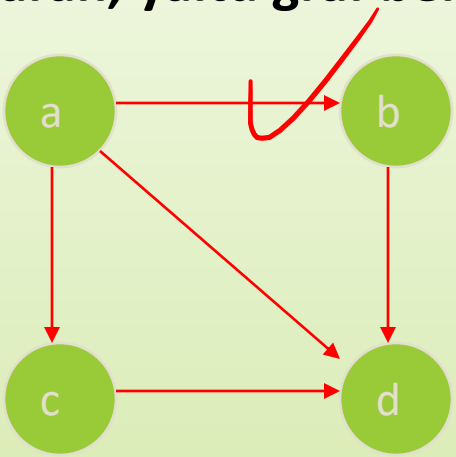
Catatan tentang BFS

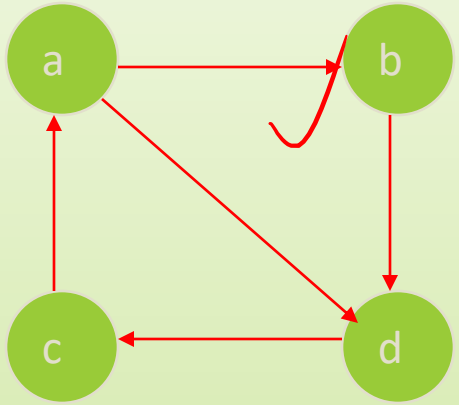
- **BFS memiliki efisiensi yang sama dengan DFS dan dapat diimplementasikan dengan grafik yang direpresentasikan sebagai :**
  - **adjacency matrices: $\Theta(|V|^2)$. Why?**
  - **adjacency lists: $\Theta(|V|+|E|)$. Why?**
- **Menghasilkan satu urutan simpul (urutan yang ditambahkan/dihapus dari antrian adalah sama)**
- **Aplikasi: sama seperti DFS, tetapi juga dapat menemukan jalur dari sebuah simpul ke semua simpul lain dengan jumlah tepi terkecil**

DAGs dan Topological Sorting

A dag: graf asiklik berarah, yaitu graf berarah tanpa siklus (berarah)

a dag





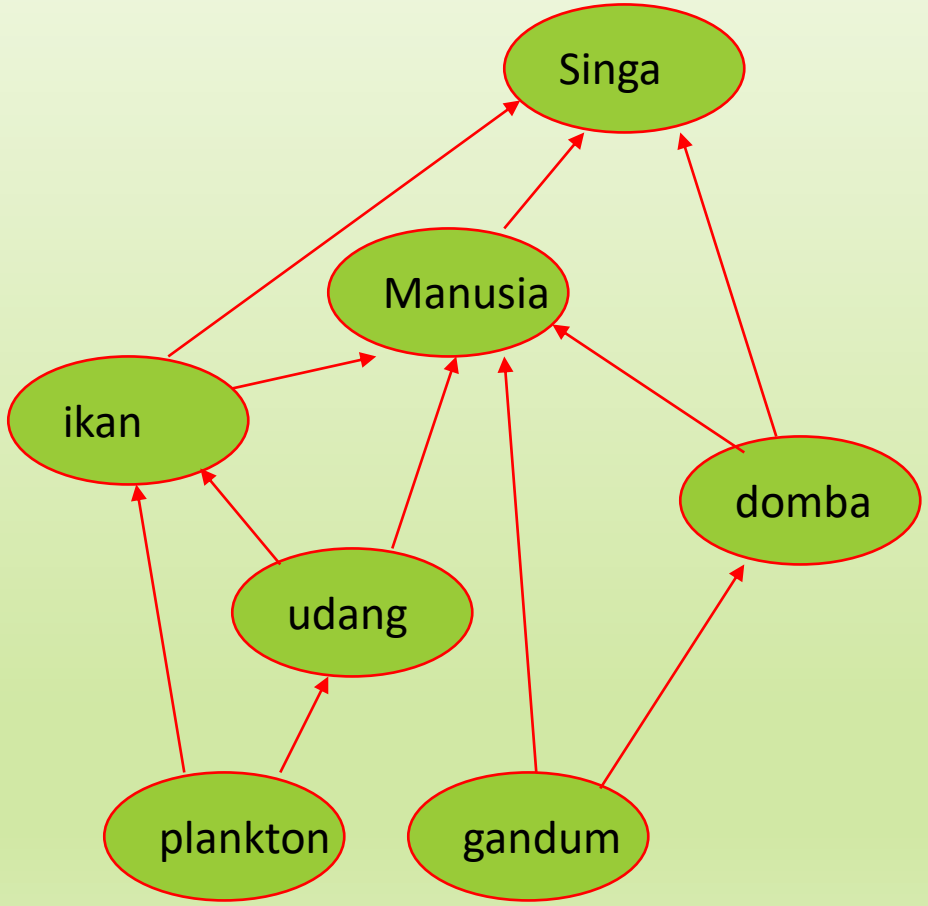
not a dag

Timbul dalam pemodelan banyak masalah yang melibatkan prasyarat kendala (proyek konstruksi, kontrol versi dokumen)

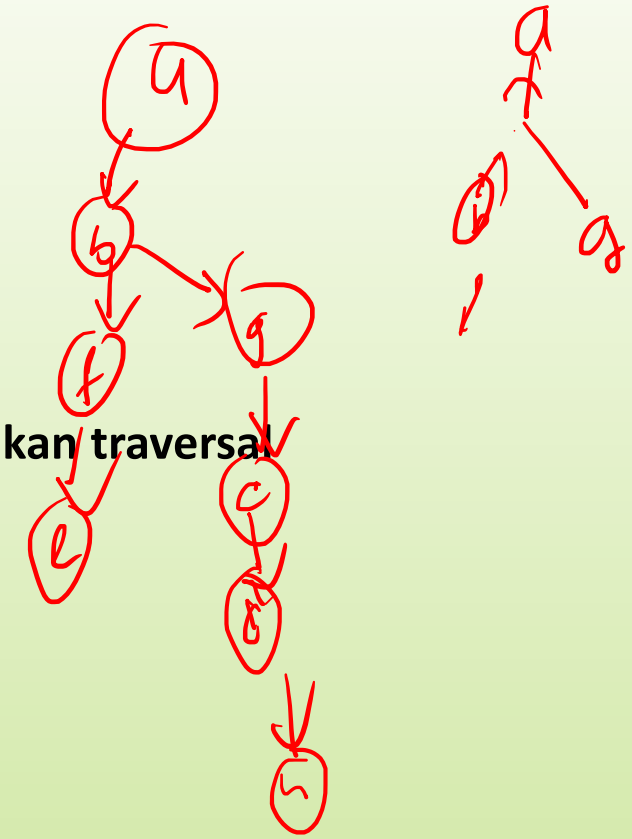
Simpul dag dapat diurutkan secara linier sehingga untuk setiap tepi simpul awalnya dicantumkan sebelum simpul akhirnya (penyortiran topologi). Menjadi dag juga merupakan kondisi yang diperlukan untuk penyortiran topologi menjadi mungkin.

Contoh Topological Sorting

Urutkan barang-barang berikut dalam rantai makanan



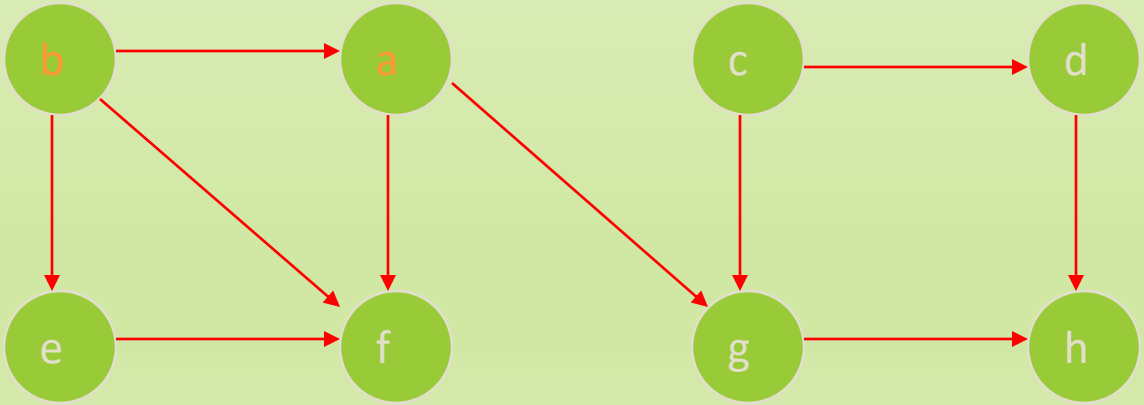
DFS-based Algorithm



DFS-based algorithm for topological sorting

- **Lakukan traversal DFS, perhatikan simpul urutan muncul dari tumpukan traversal**
- **Urutan terbalik memecahkan masalah penyortiran topologi**
- **Tepi belakang ditemui?→ NOT a dag!**

Contoh :



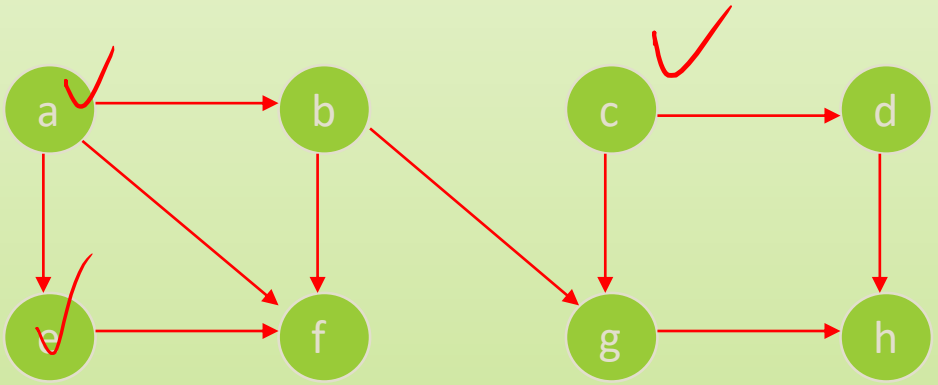
Efisiensi: Sama dengan DFS.

Source Removal Algorithm

Source removal algorithm

Identifikasi dan hapus sumber secara berulang (sebuah simpul tanpa tepi masuk) dan semua tepi yang bersinggungan dengannya sampai tidak ada simpul yang tersisa atau tidak ada sumber di antara simpul yang tersisa (not a dag)

Contoh :



"Balikkan" daftar ketetanggaan untuk setiap simpul untuk menghitung jumlah tepi yang masuk dengan menelusuri setiap daftar ketetanggaan dan menghitung berapa kali setiap simpul muncul dalam daftar ini. Untuk menghapus sumber, kurangi jumlah setiap tetangganya satu per satu.

Decrease-by-Constant-Factor Algorithms

Dalam variasi penurunan dan penaklukan ini, ukuran instans dikurangi dengan faktor yang sama (biasanya 2)

Contoh:

Pencarian biner dan metode bagi dua

Eksponen dengan mengkuadratkan

Perkalian la russe (metode petani Rusia)

Teka-teki koin palsu

Masalah Josephus

Exponentiation by Squaring

Soal: Hitung a^n di mana n adalah bilangan bulat nonnegatif

Masalahnya dapat diselesaikan dengan menerapkan rumus secara rekursif:

For even values of n $\quad a^n = (a^{n/2})^2$ if $n > 0$ and $a^0 = 1$

For odd values of n $\quad a^n = (a^{(n-1)/2})^2 a$

Recurrence: $M(n) = M(\lfloor n/2 \rfloor) + f(n)$, where $f(n) = 1$ or 2,

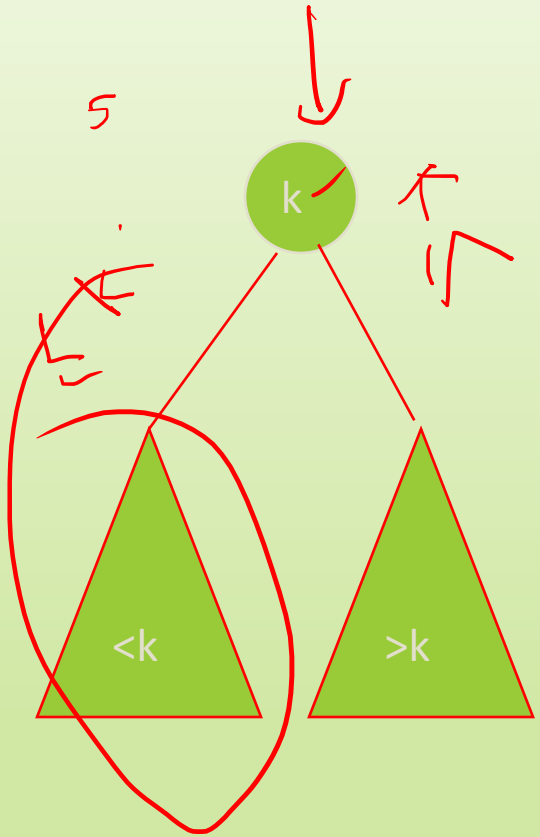
$M(0) = 0$

Master Theorem: $M(n) \in \Theta(\log n) = \Theta(b)$ where $b = \lceil \log_2(n+1) \rceil$

Binary Search Tree Algorithms

Beberapa algoritme pada BST memerlukan pemrosesan rekursif hanya pada salah satu subpohonnya, mis.,

- **Searching**
- **Insertion of a new key**
- **Finding the smallest (or the largest) key**



Searching in Binary Search Tree

Algorithm *BST*(*x*, *v*)

//Searches for node with key equal to *v* in BST rooted at node *x*

if *x* = **NIL** **return** -1

else if *v* = *K*(*x*) **return** *x*

else if *v* < *K*(*x*) **return** *BST*(*left*(*x*), *v*)

else return *BST*(*right*(*x*), *v*)

Efficiency

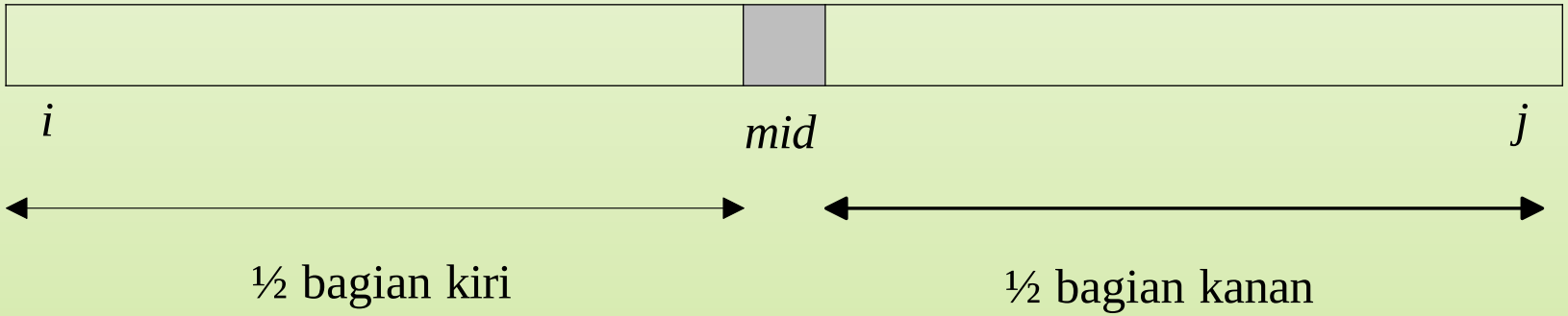
worst case: $C(n) = n$

average case: $C(n) \approx 2\ln n \approx 1.39\log_2 n$, **if the BST was built from *n* random keys and *v* is chosen randomly.**

- *Binary search*

  Kondisi awal: larik *A* sudah terurut menaik

  K adalah nilai yang dicari



Jika elemen tengah (mid) ≠ *k*, maka pencarian dilakukan hanya pada setengah bagian larik (kiri atau kanan)

Ukuran persoalan selalu berkurang sebesar setengah ukuran semula. Hanya setengah bagain yang diproses, setengah bagian lagi tidak.

```
procedure bin_search(input A : ArrayOfInteger;
                     input i, j : integer; input K : integer; output idx : integer)
Deklarasi
    mid : integer

Algoritma:
    if i > j then   { ukuran larik sudah 0}
        idx ← -1    { k tidak ditemukan }
    else
        mid ← (i + j)/2
        if A(mid) = k then   { k ditemukan }
            idx ← mid        { indeks elemen larik yang bernilai = K }
        else
            if K > A(mid  then
                bin_search(A, mid + 1, j, K, idx)
            else
                bin_search(A, i, mid - 1 , K, idx)
            endif
        endif
endif
```

- Jumlah operasi perbandingan:

$$T(n) = \begin{cases} 0 & , n = 0 \\ 1 + T(n/2) & , n > 0 \end{cases}$$

- Relasi rekursif tsb diselesaikan sbb:

T(n) = 1 + T(n/2)

= 1 + (1 + T(n/4)) = 2 + T(n/4)

= 2 + (1 + T(n/8)) = 3 + T(n/8)

= … = j + $T(n/2^j)$

Asumsi: $n = 2^j \rightarrow j = {}^2\log n$

$T(n) = {}^2\log n + T(1) = {}^2\log n + (1 + T(0)) = 1 + {}^2\log n = O({}^2\log n)$

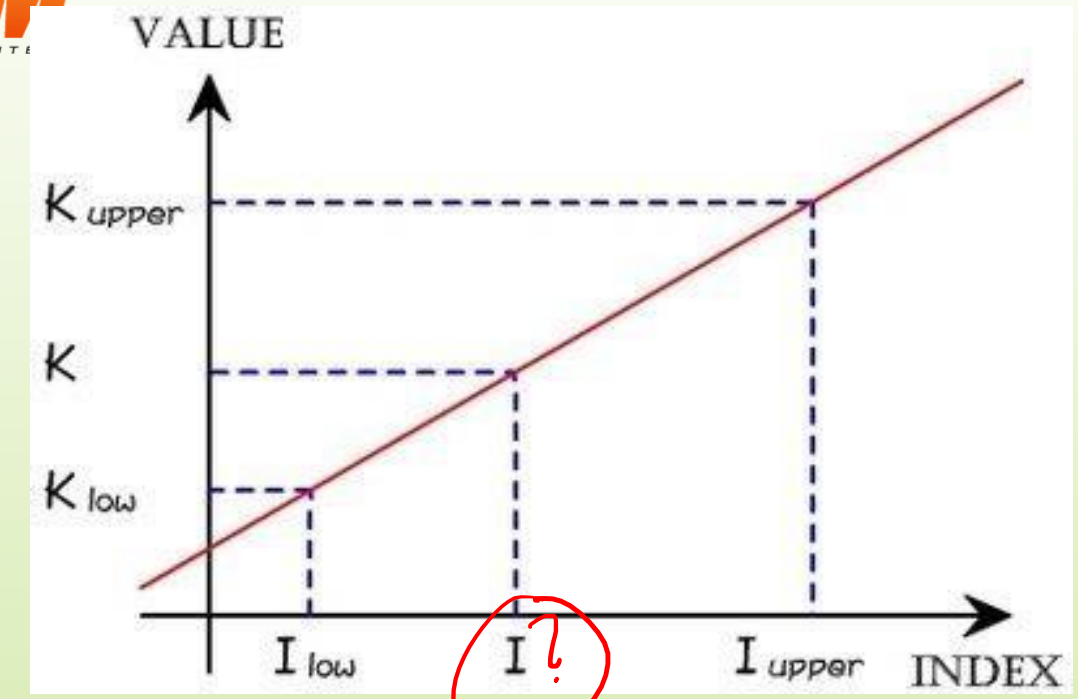
Interpolation Search

- ***Interpolation Search***

  - Analog dengan pencarian data di dalam kamus dengan cara perkiraan letak.

  - Kondisi awal: larik *A* sudah terurut menaik

    - ***K* adalah nilai yang dicari**



$$\frac{K - K_{low}}{K_{upper} - K_{low}} = \frac{I - I_{low}}{I_{upper} - I_{low}}$$

$$I = I_{low} + (I_{upper} - I_{low}) \times \frac{K - K_{low}}{K_{upper} - K_{low}}$$

```
procedure interpolation_search(input A : ArrayOfInteger;
                               input i, j : integer; input K : integer; output idx : integer)
Deklarasi
        mid : integer

Algoritma:
      if i > j then       { ukuran larik sudah 0}
          idx ← -1        { K tidak ditemukan }
      else
          mid ← i  + (j – i) *(K – A(i))/ (A(j) – A (i))
          if A(mid) = x then    { K ditemukan }
              idx ← mid         { indeks elemen larik yang bernilai = K }
          else
              if  K < A(mid  then
                  interpolation_search(A, mid + 1, j, K, idx)
              else
                  interpolation_search(A, i, mid - 1 , K, idx)
              endif
          endif
 endif
```

- Kompleksitas algoritma *interpolation search*:

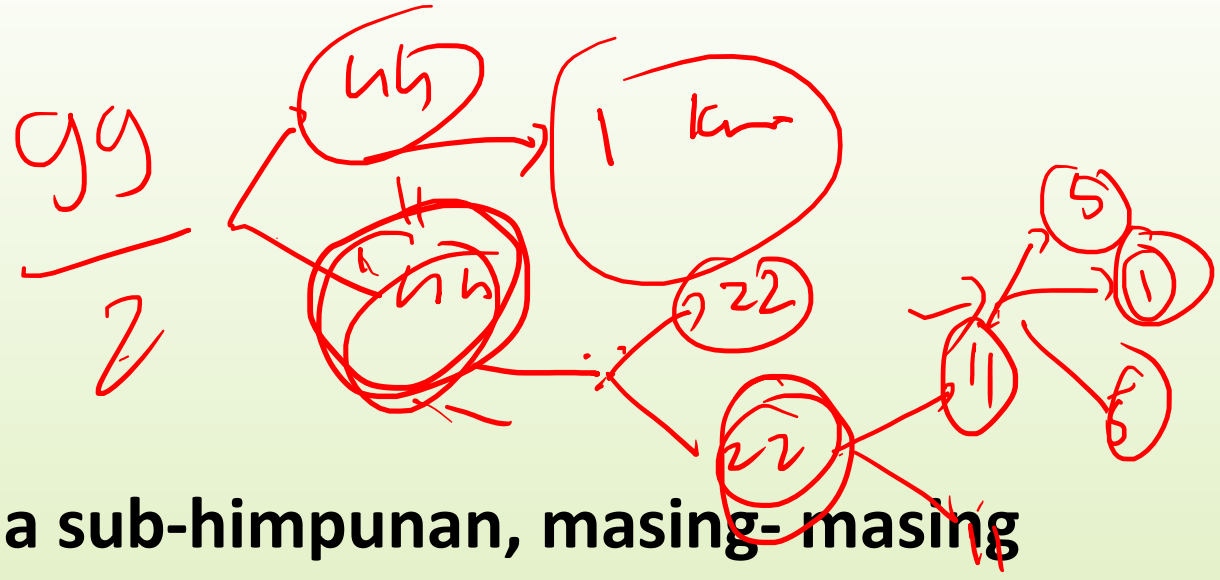
  - Kasus terburuk: $O(n)$, untuk sembarang distribusi data

  - Kasus terbaik: $O(\log \log n)$, jika data di dalam senarai terdistribusi *uniform*

Mencari koin palsu

- Diberikan *n* buah koin yang identik, satu diantaranya palsu. Asumsikan koin yang palsu mempunyai berat yang lebih ringan daripada koin asli. Untuk mencari yang palsu, disediakan sebuah timbangan yang teliti. Carilah koin yang palsu dengan cara penimbangan.

Mencari koin palsu



- Algoritma *decrease and conquer*:
  1. Bagi himpunan koin menjadi dua sub-himpunan, masing- masing $\lfloor n/2 \rfloor$ koin. Jika *n* ganjil, maka satu buah koin tidak dimasukkan ke dalam kedua sub-himpunan.
  2. Timbang kedua sub-himpunan dengan neraca.
  3. Jika beratnya sama, berarti satu koin yang tersisa adalah palsu.
  4. Jika beratnya tidak sama, maka ulangi proses untuk sub-himpunan yang beratnya lebih ringan (salah satu koin di dalamnya palsu).

- Ukuran persoalan selalu berkurang dengan faktor setengah dari ukuran semula. Hanya setengah bagian yang diproses, setengah bagian yang lain tidak diproses.

- Jumlah penimbangan yang dilakukan adalah:

$$T(n) = \begin{cases} 0 & ,n = 1 \\ 1 + T(\lfloor n/2 \rfloor) & ,n > 1 \end{cases}$$

- Penyelesaian relasi rekurens T(n) mirip seperti *binary search*:

$$T(n) = 1 + T(\lfloor n/2 \rfloor) = \ldots. = O(^2 \log n)$$

Click to edit subtitle style

Thank You !